GigaDevice Semiconductor Inc.

FATFS file system porting based on GD32 MCU

Application Note
AN065

Revision 1.0

(Aug. 2023)

Table of Contents

List of Figures

List of Tables

1. Introduction to FATFS

A file system is an organizational structure for storing and managing data on a storag e medium, including the system boot area, directories, and files. Before establishing a file system on a storage medium, it is necessary to format the storage medium to er ase the original data, and then create a new file allocation table and directory, so as to record and manage the physical address and remaining space of data storage, just like a library management system.

The file system is huge and complex, and needs to be written according to the file system format of the application, and is generally separated from the driver layer to facilitate porting. Therefore, open source file system source code is usually ported in engineering applications. FATFS is a general-purpose FAT file system for small embed -ded systems. FATFS is written based on the ANSIC language and is completely inde pendent of the underlying I/O medium. Therefore, FATFS can be easily ported to avar -iety of processors.

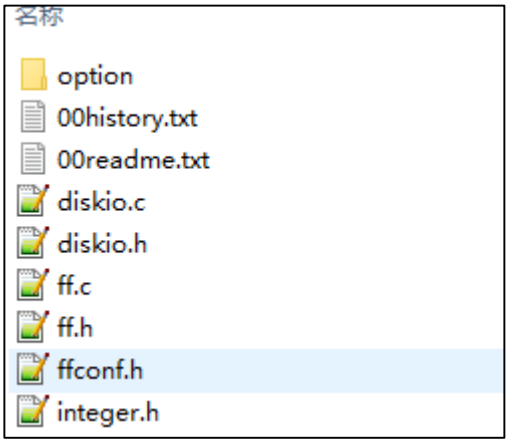
2. FATFS porting

2.1. FATFS porting platform

Based on the GD32103C EVAL development board, this paper transplants FATFS, and realizes the file management of stored data on SPI-flash (GD25Q16), on-chip flash and SD card through FATFS. The IDE platform for FATFS porting is keil4.

The FATFS file system package can be obtained from the website ***http://elm-chan.org/fsw/ff/00index_e.html*** , and the files it contains are: 1. Files related to the porting of the underlying interface hardware driver: diskio.c, diskio..h; 2. Files related to FATFS module and used to implement FAT file read/write protocol: ff.c, ff.h; 3. Files related to FATFS module configuration: ffconf.h; 4. Related to data types 5. FATFS provides folders related to external functions that support different languages: option folder, as shown in ***Figure 2-1. FATFS package file structure***. The version of the transplanted FATFS file system is R0.11a.

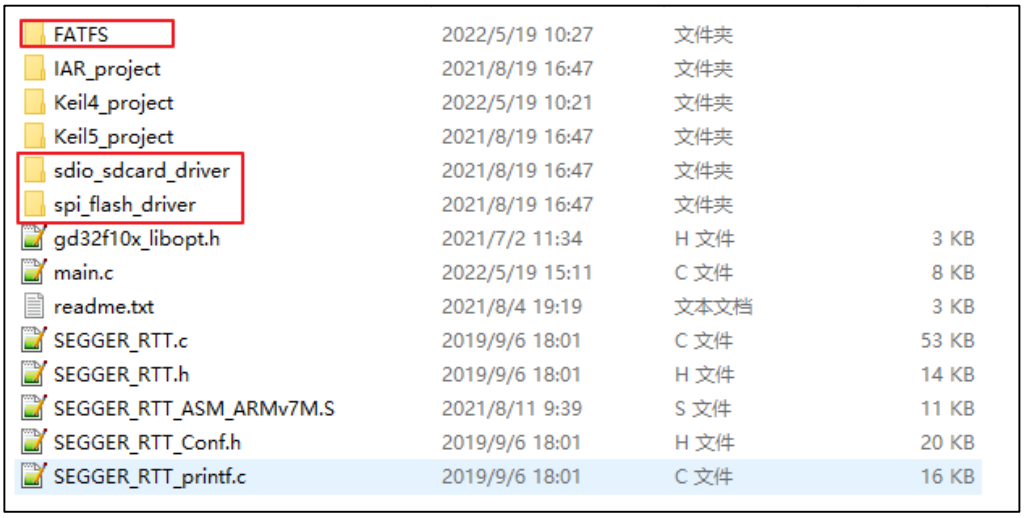
Figure 2-1. FATFS package file structure



2.2. Add FATFS source code

Name the project folder SDIO_FATFS, and add the downloaded and decompressed FATFS package under this folder, and add the SD card driver file and the SPI_flash driver file. In order to facilitate porting, it can be modified based on the 16_SDIO_SDCardTest routine in Demo_Suites. After adding the above files to the project, the contents of the folder are as shown in ***Figure 2-2. Project file***.

Figure 2-2. Project file



2.3. Modify the ffconf.h file

Open the ffconf.h file in the FATFS folder and modify the following parts:

1. Enable the file system mount function f_mkfs(), and define the value of the compiled macro _USE_MKFS as 1;
2. Modify the number of supported hardware drivers to 3. This file system migration supports three storage media (SD card, SPI_Flash, on-chip Flash), and the value of the compiled macro _VOLUMES is defined as 3;
3. Define the range of the sector size of the storage medium. According to the storage medium used in this migration, define the value of the compiled macro _MIN_SS as 512 and the value of the compiled macro _MAX_SS as 4096;

2.4. Modify the diskio.c file

Open the diskio.c file and write the underlying driver, in which disk_initialize is used for the initialization of the storage medium; disk_status is used to obtain the working status of the storage medium; disk_read is used for the read operation of the storage medium; disk_write is used for the write operation of the storage medium; disk_ioctl is used for It is used to obtain the block size and number of blocks of the storage medium; get_fattime is used to obtain the file system timestamp. This function is not used in this migration and will not be modified. Integrate the driver interface codes of SD card, SPI_Flash and on-chip Flash into diskio.c respectively. The code is modified as shown in ***Table 2-1. The code of diskio.c.***

Table 2-1. The code of diskio.c

```
#include "diskio.h"        /* FatFs lower layer API */
#include "spi_flash.h"
#include "sdcard.h"
```

```
#define   SD_CARD          0
#define   SPI_FLASH        1
#define   INTER_FLASH      2
#define   FLASH_SECTOR_COUNT   512        /*SPI_Flash SECTOR number*/
#define   FLASH_SECTOR_SIZE    4096       /*SPI_Flash SECTOR size*/
#define   FLASH_BLOCK_SIZE     1          /*smallest unit of erased sector*/
#define   SD_CARD_BLOCK_SIZE   1
#define   FMC_WRITE_START_ADDR      ((uint32_t)0x08000000U)
extern   sd_card_info_struct   sd_cardinfo;
/*-----------------------------------------------------------------------*/
/* Get Drive Status                                                      */
/*-----------------------------------------------------------------------*/

DSTATUS disk_status(
    BYTE pdrv          /* Physical drive nmuber to identify the drive */
)
{
    DSTATUS stat;

    switch(pdrv) {
    case SD_CARD :
        return 0;
    case SPI_FLASH :
        if(spi_flash_ID_read() == 0xC84015) {
            stat =   0; //initialization normal
        } else {
            stat = STA_NOINIT; //initialize not normal
        }
        return stat;
    case INTER_FLASH:
        stat =   0;
        return stat;
    }
    return STA_NOINIT;
}

/*-----------------------------------------------------------------------*/
/* Inidialize a Drive                                                    */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize(
    BYTE pdrv/* Physical drive nmuber to identify the drive */
```

```
)
{
    DSTATUS stat;

    switch(pdrv) {
    case SD_CARD:
        stat &= ~STA_NOINIT;
        return 0;
    case SPI_FLASH :
        spi_flash_config();
        return   disk_status(SPI_FLASH);
    case INTER_FLASH:
        stat =   0;
        return stat;
    }
    return STA_NOINIT;
}

/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/

DRESULT disk_read(
    BYTE pdrv,          /* Physical drive nmuber to identify the drive */
    BYTE *buff,         /* Data buffer to store read data */
    DWORD sector,    /* Sector address in LBA */
    UINT count          /* Number of sectors to read */
)
{
    uint32_t *ptrd, *btrd;
    DRESULT res;
    sd_error_enum   SD_stat = SD_OK;
    switch(pdrv) {
    case SD_CARD :
        if(count > 1) {
            SD_stat = sd_multiblocks_read((uint32_t *)buff, sector * sd_cardinfo.card_blocksize,
sd_cardinfo.card_blocksize, count);
        } else {
            SD_stat = sd_block_read((uint32_t *)buff, sector * sd_cardinfo.card_blocksize,
sd_cardinfo.card_blocksize);
        }
        if(SD_stat == SD_OK) {
            res = RES_OK ;
```

```
            } else {
                res = RES_ERROR ;
            }
            return res;

        case SPI_FLASH :
            spi_flash_buffer_read((uint8_t *)buff, sector * FLASH_SECTOR_SIZE, count * FLASH_SECTOR_SIZE);
            res = RES_OK;

            return res;
        case INTER_FLASH:
            btrd = (uint32_t *)buff;
            for(ptrd = (uint32_t *)(FMC_WRITE_START_ADDR + (sector + 47) * 2048);ptrd < (uint32_t *)(FMC_WRITE_START_ADDR + ((sector + 47) * 2048) + (count * 2048)); ptrd++) {
                *btrd = *ptrd;
                btrd++;
            }
            res = RES_OK;
            return res;
        }
        return RES_PARERR;
}

/*-----------------------------------------------------------------------*/
/* Write Sector(s)                                                       */
/*-----------------------------------------------------------------------*/

#if _USE_WRITE
DRESULT disk_write(
        BYTE pdrv,              /* Physical drive nmuber to identify the drive */
        const BYTE *buff,       /* Data to be written */
        DWORD sector,           /* Sector address in LBA */
        UINT count              /* Number of sectors to write */
)
{
        DRESULT res;
        sd_error_enum    SD_stat = SD_OK;
        uint32_t address;
        uint32_t erase_counter;

        switch(pdrv) {
        case SD_CARD :
```

```
            if(count > 1) {
                SD_stat = sd_multiblocks_write((uint32_t *)buff, sector * sd_cardinfo.card_blocksize, sd_cardinfo.card_blocksize, count);
            } else {
                SD_stat = sd_block_write((uint32_t *)buff, sector * sd_cardinfo.card_blocksize, sd_cardinfo.card_blocksize);
            }
            if(SD_stat == SD_OK) {
                res = RES_OK ;
            } else {
                res = RES_ERROR ;
            }
            return res;

        case SPI_FLASH:
            /*first erase then write*/
            spi_flash_sector_erase(sector * FLASH_SECTOR_SIZE);
            spi_flash_buffer_write((uint8_t *)buff, sector * FLASH_SECTOR_SIZE, count * FLASH_SECTOR_SIZE);
            res = RES_OK;
            return res;
        case INTER_FLASH:
            fmc_unlock();
            fmc_flag_clear(FMC_FLAG_BANK0_END);
            fmc_flag_clear(FMC_FLAG_BANK0_WPERR);
            fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
            /* erase the flash pages */
            for(erase_counter = 0; erase_counter < count; erase_counter++) {
                fmc_page_erase(FMC_WRITE_START_ADDR + ((sector + 47) * 2048) + (2048 * erase_counter));
                fmc_flag_clear(FMC_FLAG_BANK0_END);
                fmc_flag_clear(FMC_FLAG_BANK0_WPERR);
                fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
            }
            address = (sector + 47) * 2048 + FMC_WRITE_START_ADDR;
            while(address < (((sector + 47) * 2048 + FMC_WRITE_START_ADDR) + count * 2048)) {
                fmc_word_program(address, *(uint32_t *)buff);
                address += 4;
                buff += 4;
                fmc_flag_clear(FMC_FLAG_BANK0_END);
                fmc_flag_clear(FMC_FLAG_BANK0_WPERR);
                fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
            }
```

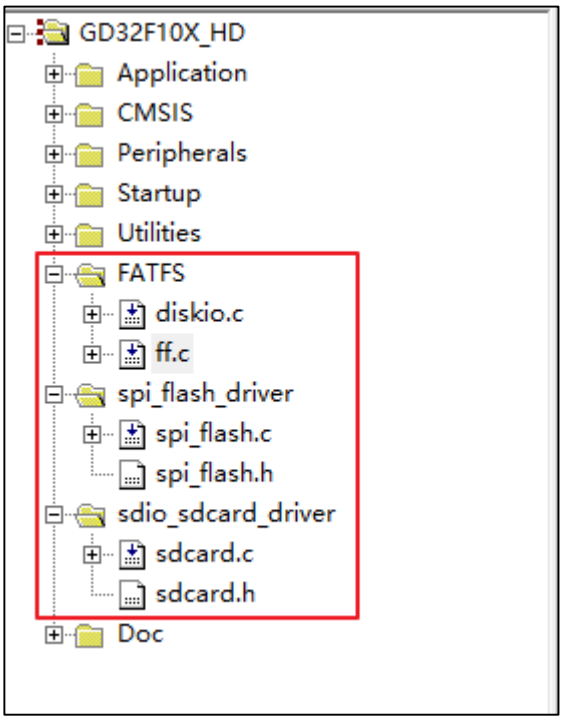
```
            fmc_lock();
            res = RES_OK;
            return res;
        }
        return RES_PARERR;
}
#endif
/*-----------------------------------------------------------------------*/
/* Miscellaneous Functions                                               */
/*-----------------------------------------------------------------------*/

#if _USE_IOCTL
DRESULT disk_ioctl(
        BYTE pdrv,              /* Physical drive nmuber (0..) */
        BYTE cmd,               /* Control code */
        void *buff              /* Buffer to send/receive control data */
)
{
        DRESULT res;

        switch(pdrv) {
        case SD_CARD :
            switch(cmd) {
            /*return sector number*/
            case GET_SECTOR_COUNT:
                *(DWORD *)buff = sd_cardinfo.card_capacity / (sd_cardinfo.card_blocksize);
                break;
            /*return each sector size*/
            case GET_SECTOR_SIZE:
                *(WORD *)buff = sd_cardinfo.card_blocksize;
                break;
            /*Returns the smallest unit of erased sector (unit 1)*/
            case GET_BLOCK_SIZE:
                *(DWORD *)buff = SD_CARD_BLOCK_SIZE;
                break;
            }
            res = RES_OK;
            return res;
        case SPI_FLASH :
            switch(cmd) {
            /*return sector number*/
            case GET_SECTOR_COUNT:
                *(DWORD *)buff = FLASH_SECTOR_COUNT;
```

```
            break;
        /*return each sector size*/
        case GET_SECTOR_SIZE:
            *(WORD *)buff = FLASH_SECTOR_SIZE;
            break;
        /*Returns the smallest unit of erased sector (unit 1)*/
        case GET_BLOCK_SIZE:
            *(DWORD *)buff = FLASH_BLOCK_SIZE;
            break;
        }
        res = RES_OK;
        return res;
    case INTER_FLASH:
        switch(cmd) {
        /*return sector number*/
        case GET_SECTOR_COUNT:
            *(DWORD *)buff = 128;
            break;
        /*return each sector size*/
        case GET_SECTOR_SIZE:
            *(WORD *)buff = 2048;
            break;
        /*Returns the smallest unit of erased sector (unit 1)*/
        case GET_BLOCK_SIZE:
            *(DWORD *)buff = 1;
            break;
        }
        res = RES_OK;
        return res;
    }
    return RES_PARERR;
}
#endif
DWORD get_fattime(void)
{
    return 0;
}
```

2.5. Add project code

Add the driver of each storage medium in keil4, and add the FATFS code. After adding, the project directory is shown in ***Figure 2-3. Project directory***.

Figure 2-3. Project directory



3. FATFS file system test

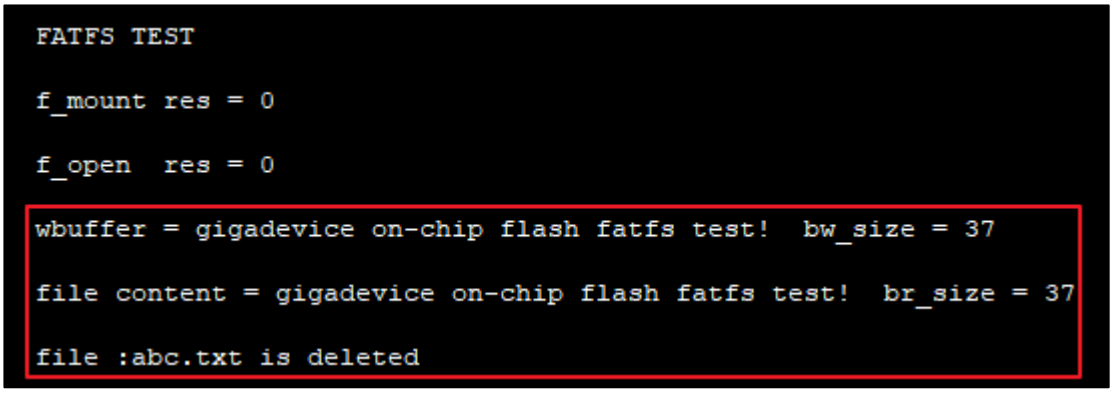
1. Use the FATFS file system to perform file addition, deletion and read/write tests on the on-chip Flash, and use the J-Link RTT Viewer to print the results. The test code is as follows:

```
void on-chip_flash_fatfs_test(void)
{
    FRESULT res;
    SEGGER_RTT_printf(0, "\r\n FATFS TEST \r\n");
    res = f_mount(&fsObject, "2:", 1);
    SEGGER_RTT_printf(0, "\r\n f_mount res = %d \r\n", res);
    if(res == FR_NO_FILESYSTEM) {
        /*creates an FAT volume on on- chip FLASH(format)*/
        res = f_mkfs("2:", 0, 0);
        SEGGER_RTT_printf(0, "\r\n f_mkfs res = %d \r\n", res);
        /*unmount file system*/
        res = f_mount(NULL, "2:", 1);
        /*mount file system*/
        res = f_mount(&fsObject, "2:", 1);
        SEGGER_RTT_printf(0, "\r\n f_mkfs 2 res = %d \r\n", res);
    }
    /*create a file enabile write and read*/
    res = f_open(&fp, "2:abc.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ);

    SEGGER_RTT_printf(0, "\r\n f_open   res = %d \r\n", res);
    if(res == FR_OK) {
        /*write data into a file*/
        res = f_write(&fp, wbuffer1, sizeof(wbuffer1), &bw_size);
        SEGGER_RTT_printf(0, "\r\n wbuffer = %s   bw_size = %d\r\n", wbuffer1, bw_size);
        if(res == FR_OK) {
            f_lseek(&fp, 0);
            /*read data from a file*/
            f_read(&fp, rbuffer, f_size(&fp), &br_size);
            if(res == FR_OK) {
                SEGGER_RTT_printf(0, "\r\n file content = %s   br_size = %d\r\n", rbuffer,
br_size);
            }
        }
        f_close(&fp);
        res = f_unlink("2:abc.txt");
        res = f_open(&fp, "2:abc.txt", FA_READ);
        if(res != FR_OK) {
            SEGGER_RTT_printf(0, "\r\n file :abc.txt is deleted \r\n");
```

```
        }
    }
}
```

The test results are shown in ***Figure 3-1. The FATFS on-chip Flash file addition, deletion, and read/write results***, indicating that the FATFS file system has successfully implemented the on-chip Flash addition, deletion, and read/write abc.txt file.

Figure 3-1. The FATFS on-chip Flash file addition, deletion, and read/write results



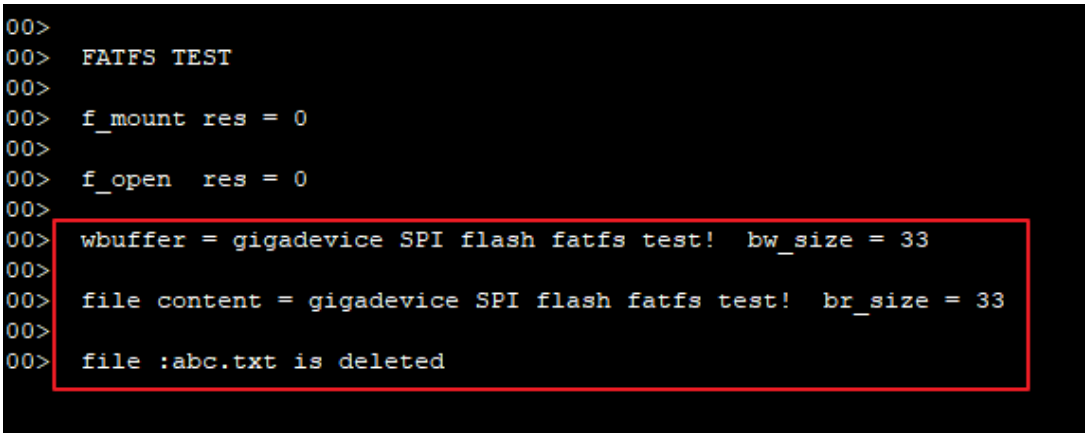
2. Use the FATFS file system to test the addition, deletion, and reading of files in SPI_Flash, and use the J-Link RTT Viewer to print the results. The test code only needs to change the drive letter in the on-chip Flash test code to SPI_Flash to test. The test code is as follows:

```
void SPI_Flash_fatfs_test(void)
{
    FRESULT res;
    SEGGER_RTT_printf(0, "\r\n FATFS TEST \r\n");
    res = f_mount(&fsObject, "1:", 1);
    SEGGER_RTT_printf(0, "\r\n f_mount res = %d \r\n", res);
    if(res == FR_NO_FILESYSTEM) {
        /*creates an FAT volume on SPI FLASH(format)*/
        res = f_mkfs("1:", 0, 0);
        SEGGER_RTT_printf(0, "\r\n f_mkfs res = %d \r\n", res);
        /*unmount file system*/
        res = f_mount(NULL, "1:", 1);
        /*mount file system*/
        res = f_mount(&fsObject, "1:", 1);
        SEGGER_RTT_printf(0, "\r\n f_mkfs 2 res = %d \r\n", res);
    }
    /*create a file enabile write and read*/
    res = f_open(&fp, "1:abc.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ);
    SEGGER_RTT_printf(0, "\r\n f_open    res = %d \r\n", res);
    if(res == FR_OK) {
        /*write data into a file*/
```

```
        res = f_write(&fp, wbuffer1, sizeof(wbuffer1), &bw_size);
        SEGGER_RTT_printf(0, "\r\n wbuffer = %s    bw_size = %d\r\n", wbuffer1, bw_size);
        if(res == FR_OK) {
            f_lseek(&fp, 0);
            /*read data from a file*/
            f_read(&fp, rbuffer, f_size(&fp), &br_size);
            if(res == FR_OK) {
                SEGGER_RTT_printf(0, "\r\n file content = %s    br_size = %d\r\n", rbuffer,
br_size);
            }
        }
        f_close(&fp);
        res = f_unlink("1:abc.txt");
        res = f_open(&fp, "1:abc.txt", FA_READ);
        if(res != FR_OK) {
            SEGGER_RTT_printf(0, "\r\n file :abc.txt is deleted \r\n");
        }
    }
}
```

The test results are shown in ***Figure 3-2. The FATFS SPI_Flash file addition, deletion, and read/write results***,and the file addition, deletion, and read/write are successful.

Figure 3-2. The FATFS SPI_Flash file addition, deletion, and read/write results



3. Use the FATFS file system to test the addition, deletion and reading of files in SPI_Flash, and use the J-Link RTT Viewer to print the results. The test code is as follows:

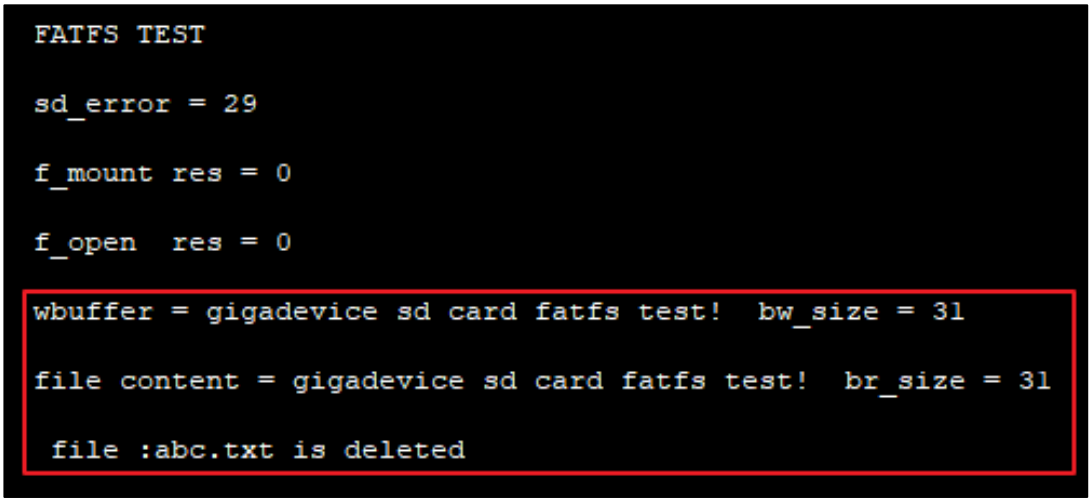
```
void sd_card_fatfs_test(void)
{
    sd_error_enum sd_error;
    uint16_t i = 5;
    FRESULT res;
    SEGGER_RTT_printf(0, "\r\n FATFS TEST \r\n");
    /* initialize SD card*/
```

```
    do {
        sd_error = sd_io_init();
    } while((SD_OK != sd_error) && (--i));
    if(sd_error == SD_OK) {
        SEGGER_RTT_printf(0, "\r\n sd_error = %d\r\n", sd_error);
    }
    /* registers/unregisters file system object to the FatFs module*/
    res = f_mount(&fsObject, "0:", 1);
    SEGGER_RTT_printf(0, "\r\n f_mount res = %d \r\n", res);
    if(res == FR_NO_FILESYSTEM) {
        /*creates an FAT volume on SD card(format)*/
        res = f_mkfs("0:", 0, 512);
        SEGGER_RTT_printf(0, "\r\n f_mkfs res = %d \r\n", res);
        /*unmount file system*/
        res = f_mount(NULL, "0:", 1);
        /*mount file system*/
        res = f_mount(&fsObject, "0:", 1);
        SEGGER_RTT_printf(0, "\r\n f_mkfs 2 res = %d \r\n", res);
    }
    /*create a file enabile write and read*/
    res = f_open(&fp, "0:abc.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ);
    SEGGER_RTT_printf(0, "\r\n f_open    res = %d \r\n", res);
    if(res == FR_OK) {
        /*write data into a file*/
        res = f_write(&fp, wbuffer, sizeof(wbuffer), &bw_size);
        SEGGER_RTT_printf(0, "\r\n wbuffer = %s    bw_size = %d\r\n", wbuffer, bw_size);
        if(res == FR_OK) {
            f_lseek(&fp, 0);
            /*read data from a file*/
            f_read(&fp, rbuffer, f_size(&fp), &br_size);
            if(res == FR_OK) {
                SEGGER_RTT_printf(0, "\r\n file content = %s    br_size = %d\r\n", rbuffer,
br_size);
            }
        }
        f_close(&fp);
        res = f_unlink("0:abc.txt");
        if(res == FR_OK) {
            SEGGER_RTT_printf(0, "\r\n    file :abc.txt is deleted \r\n");
        }
    }
}
```

The test results are shown in ***Figure 3-3. The FATFS SD card file addition, deletion,***

and read/write results,and the file addition, deletion, and read/write are successful.

Figure 3-3. The FATFS SD card file addition, deletion, and read/write results



4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Aug.11 2023

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.